**Redis复习**

**第一章:redis特性介绍**

**1. 介绍**

**Redis 是完全开源免费的，是一个高性能的key-value数据库。**

**2. 特性**

**① 速度快**

1. **存储在内存中 Redis读取的速度是110000次/s，写的速度是81000次/s**

寄存器 》 1级缓存 》 2级缓存 》 内存(每秒读取次数千万级) 》 本地硬盘(35000次) 》 远程硬盘



1. **单线程 (不用考虑多线程并发问题)**

**命令是串行执行的。在一个瞬间只执行一个命令**

1. **非阻塞IO**

**IO分为两步，第一步为数据准备 第二步为数据复制（所谓数据复制阶段就是将数据从内核复制到用户控件）**

**非阻塞就是在第一步是异步的 第二步是同步的。**

**② 数据结构多**

1. **字符串 k - v**
2. **哈希结构 k – k1 - v**
3. **集合（值唯一） k – v**
4. **有序集合（值为一） k - s - v**
5. **队列 k – v**
6. **位图**
7. **Hyperloglog**
8. **GEO**

**③ 持久化**

1. **通过持久化将内存中的数据保存在硬盘上。**
2. **方式一：RDB**
3. **方式二：AOF**

**④ 功能丰富**

1. **发布订阅（基于消息的功能）**
2. **Lua脚本（自定义命令）**
3. **事物**
4. **Pipene 流水线**

**⑤ 支持哨兵、集群、主从复制**

**⑥ 支持多种语言（php、java等语言都在使用）**

**⑦ 原子性: Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行**

**第二章: 配置**

**redis密码设置:**

**在redis的配置文件redis.conf中，Redis3.2加入的新特性，开启保护模式。即开启了protected-mode，默认值为**

**protected-mode =yes**

**是说redis只允许本机登录。**

**这样登录的前提是要设置密码，即要requirepass属性，**

**requirepass = 你设置的密码**

**如要无密码登录，就要设置protected-mode =no ，即可**

**第三章: api的使用**

**1. 其他命令**

**① keys \* (所有) | [a-d] (区间) | ph? (匹配一个) | 不要在生产环境使用**

**② dbsize 计算key的个数 | 内部有个计数器统计所以可以随便使用。**

**③ exists key 判断键是否存在 | 存在返回1 , 不存在返回0**

**④ del key1 key2 删除指定key | 返回删除的个数**

**⑤ expire key s 设置多少时间过期**

**⑥ ttl key 查看剩余过期时间| -1 代表没有过期时间 -2 是已经过期 >0 代表还有多长时间过期**

**⑦ type key 查看是什么类型| 不存在key的适合返回none**

**2. 字符串**

**① get / set / del 字符串的增删查**

**② incr / decr key 自增自减**

**incrby / decrby key n 自增自减 n 用作计数器（缺乏实践）**

**③ setnx 有则不执行,没有则添加**

**④ set key value xx 当key存在则修改,不存在则不操作。**

**⑤ setex 设置过期时间**

**⑥ mget/mset 批量获取和设置 减少网络消耗。**

**比如: 10条命令执行 需要传输10次 而这个一次行都带过去了。然后在服务器在依次执行。**

**⑦ getset 设置新的值并返回旧的值。**

**补充 :**

**Strlen key 统计字节**

**Getrange 截取字符串**

**Setrange 替换字符串**

**Append key value 拼接字符串到已有字符串后面 | 返回拼接后的字节长度**

**Incrbyfloat key n 自增浮点数+n**

**Decrbyfloat key n == incrbyfloat key –n redis中没有decrbyfloat这个。可以用后面的代替**

**3. 集合(值唯一)**

**① sadd/srem key value [value1,value2…] 基本操作,添加和删除**

**如: sadd key 1 2 3 4 | srem key 1 2 3**

**② scard key 统计元素的个数 （统计）**

**③ sismember key valued 判断valued 是否在集合当中 （判断）**

**④ srandmember key count 随机取出count个元素 （查询操作）**

**⑤ spop key 随机弹出一个元素 （删除操作） 抽奖**

**⑥ smembers key 获取所有的元素 （查询操作） 谨慎使用 可以用游标分段查询 sscan**

**集合间的操作:**

**Sdiff key1 [key2…] 取差集 （不同好友）**

**Sinter key1 [key2…] 取交集 （共同好友）**

**Sunion key1 [key2…] 取并集**

**S(diff|inter|union)store distkey key1 [key2…] 将结果存储在键destkey中**

**4. 有序集合**

**集合结构图:**

|  |  |  |
| --- | --- | --- |
| Value | Score(不唯一) | Value（唯一） |
| Name | Wuchong |
| Age | 20 |

**① zadd key score value （添加）**

**② zrem key value （删除）**

**③ zscore key value （查询指定值的分数）**

**④ zrank key value （查询指定值的排名）zrevrank key value （倒序查询指定值的排名）**

**⑤ zcard key （获取元素个数） zcount key start end （获取区间元素的个数）**

**⑥ zincrby key n values （给指定值增加分数n）**

**⑦ zrange key start end [withscores] (查询区间内的值) 想要连分数一起查要带上withscores**

**zrevrange key start end [withscores] (倒序查询区间内的值) 想要连分数一起查要带上withscores**

**⑧ zrangebyscore key start end [withscores] (****查询分数区间的值)**

**zrevrangebyscore key end start [withscores] （倒序查询分数区间的值)**

**⑨ zremrangebytrank key start end 删除指定排名内的元素（排名区间删除）**

**⑩ zremrangebyscore key start end 删除指定分数区间的元素（分数区间删除）**

**有序集合间操作:**

**Zinterstore** **newKey number key1 [key2…] [weights] 取交集**

**number 代表要计算几个集合的交集、newkey保存计算结果的键。**

**Zunionstore newKey number key1 [key2…] [weights] 取并集**

**5. hash**

**结构示意图:**

|  |  |  |
| --- | --- | --- |
| Value | Field | Value |
| Name | Wuchong |
| Age | 20 |

**① hget/hset/hdel 基本操作。**

**如: hget / hdel key field获取或删除一个filed**

**Hset key fild value 添加**

**② hexists key field 判断field是否存在**

**这时候就有疑问了不是有了一个exists来判断键的吗？怎么又来一个?**

**答: 这个是判断哈希内部的filed是否存在的**

**③ hlen key 获取key对应的filed的数量**

**⑤** **hmget / hmset 批量获取和设置**

**⑥ hincrby key filed n 给某个filed自增**

**⑦ hsetnx key vlued 当****不存在的时候才设置该filed的值。**

**⑧ hincrbyfloat key vlued 为指定field的值加上一个浮点数增量**

**不建议使用的:时间复杂度都是On**

**hgetall key 获取所有的 field -> value 对。**

**hvals key 获取所有的value**

**hkeys key获取所有的filed**

**6. 队列**

**① rpush / lpush / rpop / lpop 基本操作 添加可以一次添加多个**

**② llen key 统计长度**

**③ linsert key before|after value newValue 在指定value前后插入newValue 时间复杂度为On**

**④ lrem key count value 删除列表中指定value的元素**

**Count :**

**> 0 从左向右 删除count个与value相等的值**

**< 0 从右向左 删除count个与value相等的值**

**= 0 删除所有与value相等的值**

**⑤ ltrim key start end (删除)保留在区间的队列**  **| start 开始位置（从零开始） end 结束位置 前包后也包 On**

**⑥ lrange key start end 截取队列**

**|start 开始位置（从零开始）| end（可以从-1开始）结束位置| 前包后也包 On | 可以做分页**

**⑦ lindex key index 获取指定元素 | index 下标**

**⑧ lset key index newValue (更新) 修改对应下标的值。**

**阻塞使用（消息队列可以使用）:**

**Blpop key timeout 移除并获取列表左边第一个元素，如果列表没有元素会阻塞直到等待超时或可弹出元素为止**

**Brpop key timeout移除并获取列表右边第一个元素，如果列表没有元素会阻塞直到等待超时或可弹出元素为止**

**Timeout: 单位为秒 当 = 0 代表不阻塞**

**使用口诀:**



**7. API解读:**

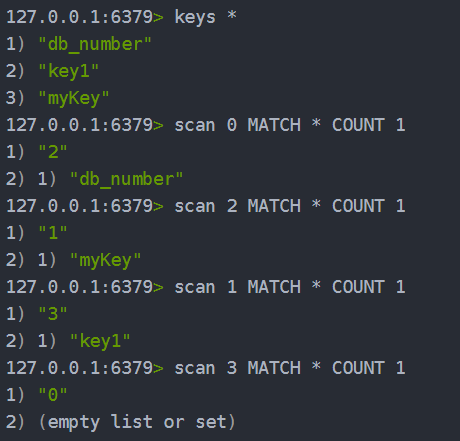
**Scan：**

**介绍 : 通过游标的方式分次获取数据，不会阻塞进程。**

**用法 : scan 游标 MATCH key count limit / key –> 过滤键名 / limit 获取的条数**

**Scan 0 match swoole\* count 3**

**游标为什么飘忽不定? 为什么不用 1 2 3 4 这种方式**

**答:** 

**我们redis中有三个key ，每次只遍历一个，游标顺序为 0 –> 2 -> 1 -> 3**

**这个游标顺序 等价于 == 00 –> 10 -> 01 -> 11**

**这时候发现: 游标增加是以二进制的方式增加的。并是高位+1; 这样对于数据扩容和缩容是非常好的。**

**详情: https://blog.csdn.net/K\_Ohaha/article/details/82990546**

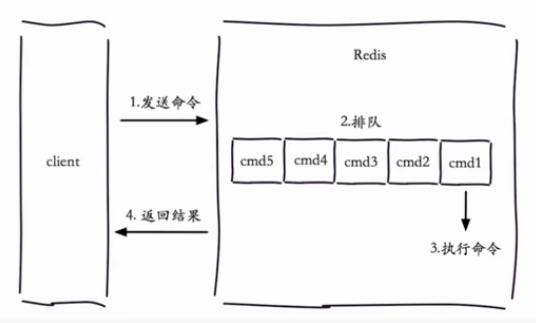
**第四章:扩展功能使用**

**1. 慢查询**

**慢查询是以队列形式保存在内存中的，默认慢查询的阈值为10000微秒 队列长度128**

**① 声明周期:（声明周期一定要理解，遇到问题就要想一下这个声明周期）**

**客户端发送命令 –> 单线程排队处理-> 执行命令 –> 返回结果**



**当客户端发生超时的时候四个阶段都可能是其导致因素。但是，慢查询这个概念只存在于第三个阶段,**

**比如执行 keys \* 就可能导致慢查询导致阻塞超时。**

**② 两个配置:**

**Slowlog-max-len 慢查询队列的长度。**

**运维经验:**

**因为我们是依靠慢查询记录来分析对应的命令。**

**所以这个长度应该设置的长一点 推荐1000**

**Slowlog-log-slower-than 慢查询的阈值。单位微妙。 <0 则不记录 =0 全部记录**

**运维经验:**

**根据QPS的大小来设置阈值。**

**因为有时候QPS非常大执行时间1毫秒就已经是很慢的了**

**所以Qps大的时候建议将阈值设置的小一点。**

**③ 三个命令**

**Slowlog get n 获取慢查询n条记录**

**运维经验:**

**定期通过这个命令将慢查询记录持久化起来。这样方便我们对历史久远的慢查询分析。**

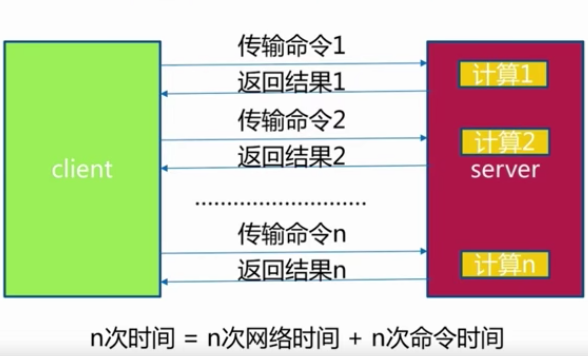
**Slowlog len 统计慢查询的总条数**

**Slowlog reset 清空慢查询**

**2. pipeline(流水线)**

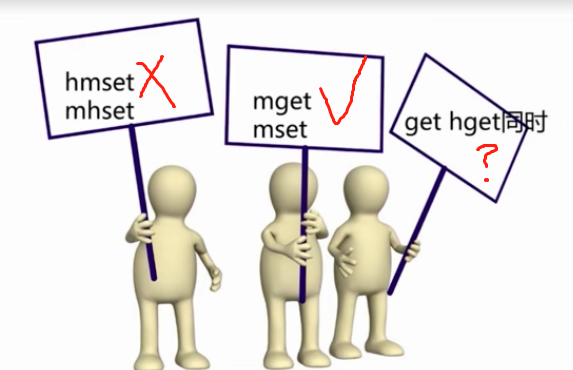
**① 为什么用流水线?**

**当我们执行N条命令的时候:**



**从上图我们可以看出，有N条命令就会有N次网络时间，我们前面说了 网络传输 也是导致慢查询的一个阶段。**

**那有人就问,怎么不用mset 和 mget呢?**



**从上图我们知道 我们进行N次set/get的字符串操作是可以用mget和mset，**

**但是当我们用hset/sadd 或者 各种命令混杂时候就没办法一起传输了。**

**那怎么办吗？**

**这时候流水线帮助了我们。**

**② 流水线（管道）的工作原理流程介绍**

**当开启管道后,命令都会被存入到缓冲区,当缓冲区满的时候或者当执行exec的时候,**

**会将缓冲区中的命令一次性发送到服务器端,服务器端收到命令后按照顺序依次执行,**

**当所有命令未全部执行完的时候,一些执行完的结果会先存入到缓存中,等所有的全部**

**完成后则返回结果给客户端。打包开始->缓存->打包结束->发送->执行->返回执行结果。**

**理解管道实现方式:**

**Pipeline 实现的原理是队列格式,先进先出的特性保证了命令的先后执行顺序。**

**理解管道的注意点:**

**不能一次传太多命令:**

**因为服务端需要处理完全部的命令才会返回,而在这之前的一些执行结果都会**

**存储在内存中占用内存资源，如果太大的话在发送的时候还可能造成网络堵塞。**

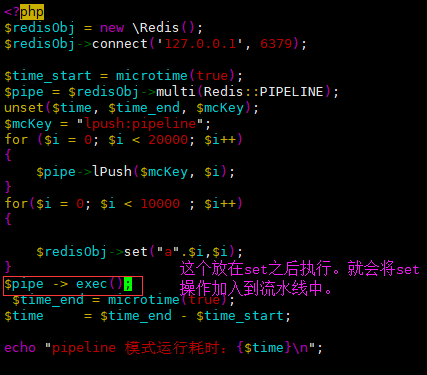
**理解管道的独占链接一说**

**因为管道是一次行发送。在执行exec()之前是独占这条链接的。**

**所以在执行pipeline前想要实现其他操作，建议再次建立客户端对服务端的链接。**

**例如:**

 **图一**

 **图二**

**图二执行时间为:** 0.057045936584473

**图一执行时间为:** 0.18010497093201

multi() 返回Redis实例并进入多模式。一旦处于多模式，所有后续方法调用将返回相同的对象，直到exec() 被调用为止

**由此可知，当pipeline为执行完的时候 所有的其他操作也会加入到管道中。所以想要执行其他操作。**

**一: 在管道关闭后执行，二: 新建立连接执行。**

**理解管道是非原子性的。**

**管道执行的是多个命令， pipeline并不是表达”所有命令都一起成功”的语义，**

**管道中前面命令失败，后面命令不会有影响，继续执行。执行失败，**

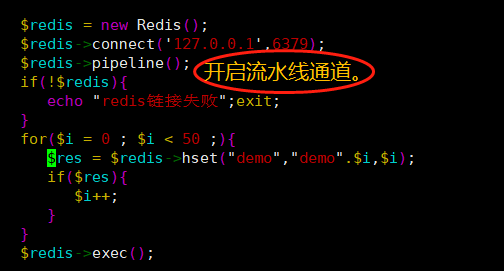
**将会在此后的相应结果集中得到信息**

**③ 流水线的用法及效率**

**因为redis执行命令的效率是微妙级别的，而网络传输消耗的时间就需要我们控制，**

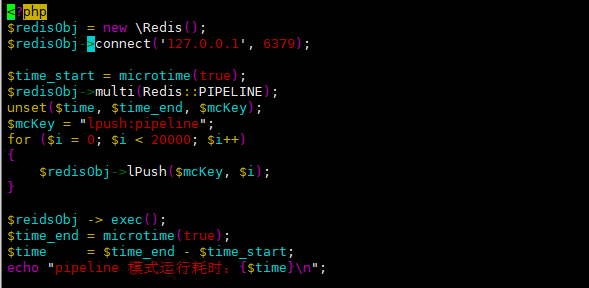
**流水线的作用就是将多次网络传输合并成一次。当然这个合并后的大小也不要太大。\**

**php用法:**



**当不调用exec()则不会发送给服务器的。**

**或者:**



**④ 问题建议**

1. **注意每次pipeline携带的数据量**

**不要太大哦。要不然有可能造成网络堵塞。**

1. **Piipeline 每次只能作用于一个节点上 （需要会集群才能理解）**
2. **M操作和 pipeline 的区别。**

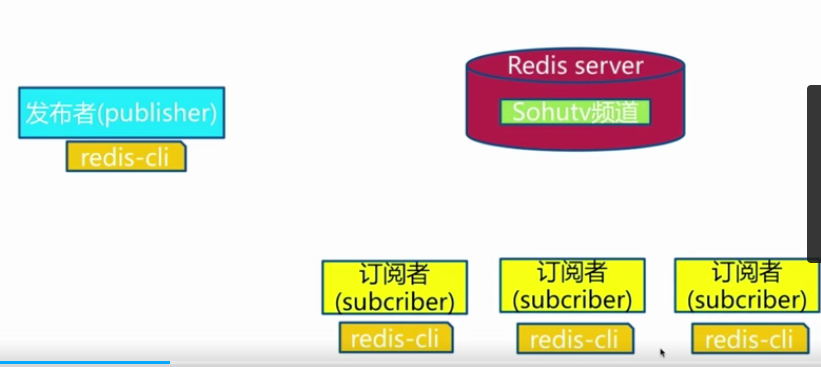
**Mset/和mget 都是原子性操作，**

**Pipeline 不是原子性的**

1. **Redis事务了解。（看后面的视频是否讲到）**

**3. 发布订阅（消息队列）**

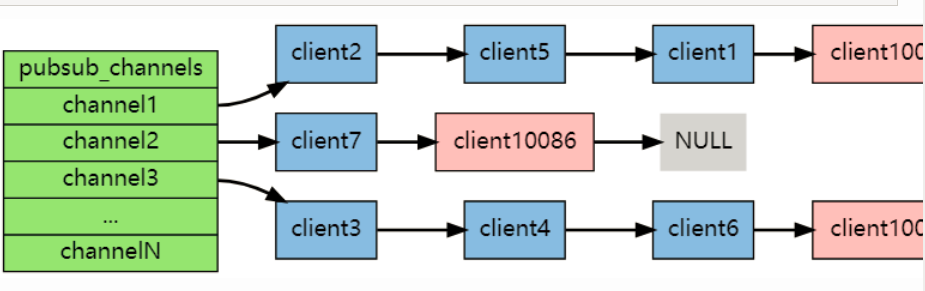
**① 发布订阅原理及流程**



**发布订阅分3个角色: 发布者、订阅者、频道。**

**在这3个角色中，发布者和订阅者都是 客户端 而 频道为 服务端。**

**整体流程就是。发布者在频道上面发送消息，订阅者如果订阅了该频道则会收到对应消息。**



**其实现方式是: 当通过subscribe订阅频道后redis会在内部维护一个k-v格式的表。**

**K(键)代表频道的名称 V(值)的数据格式是一个链表里面存放订阅该频道的所有用户。**

**所以,subscribe就是将订阅的客户存入到对应的链表中。而执行publish的时候，**

**就是将对应的链表循环遍历发布消息。**

**值得注意的是发布订阅是不存在消息堆积的，也就是看不到历史消息。**

**② 发布订阅 php及命令行使用**

**命令行:**

**Publish channel message 发布消息**

**Subscribe [channel] 订阅**

**Unsunbscribe [channel] 取消订阅**

**Psubscribe [channel]** **系列频道订阅 如: psubscribe v\* 代表订阅v开头的一系列频道**

**Punsubscribe [channel] 系列频道取消订阅**

**Pubsub channels 查看至少有一条订阅者的频道列表**

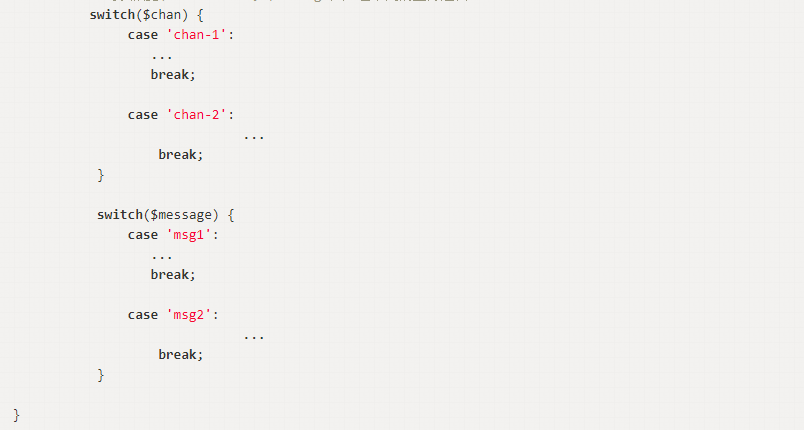
**Pubsub numsub [channel] 查看系列频道的订阅者数量**

**（这个命令返回的不是订阅模式的客户端的数量， 而是客户端订阅的所有模式的数量总和）**

**Php实现:**

**https://segmentfault.com/a/1190000020385114?utm\_source=tag-newest**





**③ 发布订阅的优缺点**

**缺点: 1. 数据传输不可靠: 当客户端断连的时候是收不到消息的。这时候就会丢失断连期间的数据。**

**2. 会阻塞后面代码执行。（待解决）;**

**优点: 简单高效，适用于一些日常需求。**

**④ list实现消息队列原理及流程**

**通过 BLPOP + lpush 实现 ( https://www.cnblogs.com/kellynic/p/9952386.html )**

**Blpop 是一个对队列阻塞弹出的一个命令。当队列中有值的时候不阻塞直接弹出。**

**当没有的时候则等待队列中压入值。W**

**多个客户端执行Blpop且队列中只有一个值的适合,那么只有一个客户端会得到这个值。**

**⑤ 发布订约（消息队列）使用场景**

**1. Pub/Sub模式比较常见的应用还是解耦两个系统吧.**

**比如我有两个程序，可能是不同编程语言构建的，可能是在不同的机器上运行的，那么利用这个模式，可以比较简单地实现这两个程序之间的通讯。**

**消息队列，就是用来解耦系统的，以消息生产者和消息消费者的角色来定义两个系统。**

**举个例子，假如V2EX有个徽章的功能，当你抢沙发超过10次时，能获得一个「抢沙发小能手」的徽章。那么一般来说，代码会这么写，先创建这条回复，然后检查是否为该主题的第一个回复，如果是，对用户抢到沙发次数加1，最后判断用户抢到沙发次数是否大于10次，如果是则发放徽章。**

**那么，当回复里包含的逻辑越多时（比如还要通知被回复者），回复这个功能的代码会越复杂。此时如果我们采用消息队列，那么可以只是简单地创建回复记录，然后\*\*发布一个回复被创建的消息\*\*到消息队列。而检查是否发放徽章的\*\*消息订阅程序\*\*，会收到前面创建的这条消息，然后去做相应的检查逻辑。**

**通过这样解耦，添加回复这块代码会变得清晰，因为这里只负责创建这条记录，并发布这个消息出去，无须关心还有哪些逻辑。而处理剩下逻辑的部分，也相当清晰，因为一个消息订阅者只处理一个逻辑。单一的逻辑非常容易进行单元测试，但是这样同时会让系统分离成好多部分，变得复杂，故只有当你的逻辑相当复杂，并且需要提高扩展性，获得更好的性能的时候才会用到这个模式**

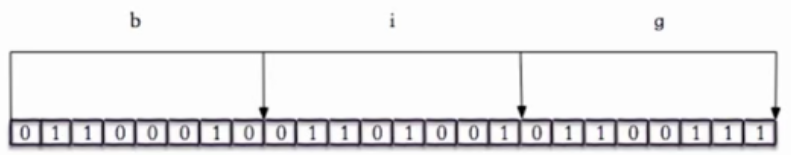
**2.**

**4. bitmap（位图）**

**① 原理及流程**

**因为redis就是对二进制的操作，而位图就是直接对二进制操作。**

**当你set demo a 的时候是这样存的, 字符a先转ASCLL码,然后用ASCll码转对应的二进制。如下图:**



**位图本质也是一个二进制字符串,也可以看做一个二进制数组。他的单位是bit。**

**因为二进制只有 0 和 1 所以位图也多使用在状态的统计分析。**

**https://www.jianshu.com/p/ddc920d3e6e0**

**② 命令及php用法**

**Setbit key offset value 设置offset下标上的值。**

**Getbit key offset 获取offset下标上的值**

**Bitcount key 统计key下面被设置为1的数量**

**Bitpos key targetbit [start end] 获取指定范围第一个等于targetbit的位置。**

**BITOP operation destkey key [key …] 取n个key的交集、并集、非、异或。Operation = or、and、not、xor**

**③ 使用场景**

**④ 问题**

[**https://www.bbsmax.com/A/Vx5MXqAvJN/**](https://www.bbsmax.com/A/Vx5MXqAvJN/)

[**https://www.jianshu.com/p/36a7775b04ec**](https://www.jianshu.com/p/36a7775b04ec)

[**https://www.jianshu.com/p/ddc920d3e6e0**](https://www.jianshu.com/p/ddc920d3e6e0)

[**https://mp.weixin.qq.com/s?\_\_biz=MzI2NzMyMDQ0NQ==&mid=100000211&idx=1&sn=1432a6e5ad7c1d76540ace8d3bfd69d7&chksm=6a81ecc15df665d7cd92edff23d6a0da3dbaa6a8044e32f7a0544c86bfc286831b4ba6dd719c#rd**](https://mp.weixin.qq.com/s?__biz=MzI2NzMyMDQ0NQ==&mid=100000211&idx=1&sn=1432a6e5ad7c1d76540ace8d3bfd69d7&chksm=6a81ecc15df665d7cd92edff23d6a0da3dbaa6a8044e32f7a0544c86bfc286831b4ba6dd719c#rd)

[**https://blog.csdn.net/zhang\_referee/article/details/88605052**](https://blog.csdn.net/zhang_referee/article/details/88605052)

[**http://www.360doc.com/content/19/0610/13/33093582\_841522174.shtml**](http://www.360doc.com/content/19/0610/13/33093582_841522174.shtml)

**https://www.cnblogs.com/liujiduo/p/10396020.html**

**5. hyperloglog**

**6. geo**

**https://www.jianshu.com/p/ddc920d3e6e0**

**第四章:redis持久化**

**1.**

**第五章:redis复制原理及优化**

**1.**

**第六章:redis 哨兵**

**1.**

**第七章:redis 集群**

**1.**

